ON ALGORITHMS FOR FINDING ALL CIRCUITS OF A GRAPH

by

Prabhaker Mateti and Narsingh Deo

June 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

On Algorithms for Finding all Circuits of a Graph*

Prabhaker Mateti

Narsingh Deo**

June 1973

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation (Grant GJ-31222).

**Department of Electrical Engineering, Indian Institute of Technology, Kanpur, India.

ACKNOWLEDGMENT

We thank Professor Jurg Nievergelt for his encouragement to write the report.

The second author is grateful to the University of Illinois for the opportunity to visit the Department of Computer Science.

We are grateful to the National Science Foundation for the financial support under Grant GJ-31222.

ABSTRACT

An efficiency analysis of all known algorithms for finding all circuits of a graph is made, and bounds on the computation time are derived. Best algorithms are exhibited. The vector-space method of circuit finding is discussed at length. Proven is the fact that the graphs K_3, K_4, K_4-x and $K_{3,3}$ are the only ones which have 2^{μ}-1 circuits (μ is the nullity of the graph), or, equivalently, have no two edge-disjoint circuits. A class of graphs, circuit-graphs, derivable out of a graph are defined. If an efficient algorithm for the enumeration of all connected, induced subgraphs is known, these circuit-graphs can advantageously be used in the generation of circuits.

1. Introduction

In many applications of graph theory, it is desired to obtain a list of all circuits of a graph. Examples are program optimization, program segmentation, testing graphs for isomorphism, signal flow-graph theory, etc. (See Prabhaker (1972)). We will not, however, dwell on the applications here. The problem considered is: Given an undirected graph (for definitions of the terms, see below), find all its circuits. The analogous problem for a digraph is to find all its dicircuits. The two problems have much in common and we consider them in parallel. In the process of looking for a better algorithm, we analyzed and compared all known algorithms for finding all circuits. To avoid misunderstanding we will define our terminology.

A graph G is a triple $<V,E,f>$ where V is a finite set of vertices, E a finite set of edges, and f is a function. A graph is either directed (a digraph) when $f:E \rightarrow V \times V$, or is undirected when $f:E \rightarrow \{\{u,v\} | u,v \in V\}$. Observe that this definition permits the graph to have parallel edges (two edges e_1 and e_2 are parallel if $e_1 \neq e_2$ and $f(e_1) = f(e_2)$) and self-loops (an edge e is a self-loop if for some vertex v, $f(e) = (v,v)$ for digraph or $f(e) = \{v\}$ for undirected graph). A graph G is simple if it does not have either self-loops or parallel edges. An edge e of a digraph is said to be incident out of vertex v_1, and incident into a vertex v_2 if $f(e) = (v_1,v_2)$. The edge e is incident with both vertices v_1 and v_2. The in-degree of a vertex v of a digraph is the number of edges incident into v, and the out-degree of v is the number of edges incident out of v. An edge e of an undirected graph is incident with vertices v_1 and v_2 if $f(e) = \{v_1,v_2\}$. The degree of a vertex v of an undirected graph is the number of edges incident with this vertex, self-loops being counted twice.

Two vertices v_1 and v_2 are adjacent if there is some edge e so that $f(e) = (v_1,v_2)$ or (v_2,v_1). An edge-sequence from vertex v_0 to v_k in a digraph is an alternating sequence of vertices and edges, $v_0e_1v_1e_2v_2\cdots v_{k-1}e_kv_k$, such that for $1 \leq i \leq k$, $f(e_i) = (v_{i-1},v_i)$. This edge-sequence is said to pass through the vertices v_i, $0 < i < k$. An edge-sequence is simple if it does not pass through a vertex more than once. A dipath from v_0 to v_k is an edge-sequence from v_0 to v_k where all vertices are distinct, except possibly that $v_0 = v_k$. When $v_0 = v_k$, the dipath is called a dicircuit. The length of a dipath is the number of edges in it. Since the vertices can be determined knowing the edges by means of the function f, vertices may be deleted from a representation of an edge-sequence. Also, since we will be considering an edge-sequence as a subgraph, the order of these edges is unimportant and an edge-sequence (and hence a dipath and dicircuit) is simply a set of these edges of the digraph.

By deleting an edge e from a graph $G = \langle V,E,f \rangle$ we obtain a new graph $G' = \langle V,E',f' \rangle$ where $E' = E - \{e\}$ and f' is a restriction of f to E'. By deleting a vertex v from a graph G, we obtain a new graph $G' = \langle V',E',f' \rangle$ where $V' = V - \{v\}$ and $E' = \{e \in E \mid f(e) \neq (v,u) \text{ or } (u,v) \text{ for } u \in V\}$ and $f':E' \to V' \times V'$ is a restriction of f to E'. By ignoring the direction of an edge e we mean that the ordered pair $f(e) = (u,v)$ be considered as a set $\{u,v\}$. The corresponding definitions for an undirected graph follow if the directions of the edges are ignored.

An undirected graph is connected if there exists a path between every pair of vertices. A vertex is a cut-vertex if it lies on every path between a pair of vertices. A connected undirected graph is separable if it has at least one cut-vertex. A maximal, connected subgraph of a graph is called a component of the graph. A maximal, nonseparable subgraph of a graph is called

a block of the graph. A directed graph is strongly-connected if there exists a dipath between every pair of vertices. A maximal, strongly-connected subgraph of a digraph is called a fragment. A spanning tree of a connected undirected graph is a maximal subgraph which has no circuits. The unique circuit obtained by adding a non-tree edge to the spanning tree is called a fundamental circuit (with respect to that spanning tree).

Let n be the number of vertices and m the number of edges. We shall refer to an edge-sequence of length k as a k-sequence. Similarly, we use k-dipath and k-dicircuit.

2. Various Methods of Circuit Finding

The algorithms for finding all circuits of a graph fall into four broad classes depending on the underlying approach.

1. Circuit vector space for an undirected graph (Maxwell and Reed (1965), Welch (1966), Rao and Murti (1969))

2. Search algorithms (Char (1968), Chan and Chang (1969), Roberts and Flores (1966), Floyd (1967), Tiernan (1970), Weinblatt (1972))

3. Powers of adjacency matrix (Ponstein (1966), Kamae (1967), Yau (1967), Danielson (1968))

4. Edge-digraph (Cartwright and Gleason (1966)).

The details of these algorithms often obscure the basic ideas behind them. To gain insight, and to be able to point out specific sources of inefficiencies, we give a general description of these four methods of circuit finding. Some new results are also given; for example, it is shown that there are only four undirected graphs whose circuit vector space consists of all circuits (ignoring the null element). We also show that there are classes of graphs for which the ratio of number of circuits to all the vectors in the circuit vector space

goes asymptotically to zero, as the number of vertices increases. Unifying these observations, we suggest improvements. A new class of undirected graphs -- circuit-graphs of an undirected graph -- are defined which aid the process of finding circuits in undirected graphs. Some properties of these graphs are explored.

In the rest of the paper, by the term "graph" we shall refer simultaneously to undirected, as well as directed graphs. The term "circuits" will also be used likewise. Should there be a possibility of confusion, an explicit mention will be made as to whether we are discussing the directed or undirected graphs.

2.1 Circuit Vector Space of an Undirected Graph.

It is well known that the set of all circuits and edge-disjoint unions of circuits of an undirected graph form a vector space over GF(2). The vector operation is the "ring-sum" operation (see Harary (1969) or Deo (1974)). A non-null vector in this circuit vector space is called a circ. To obtain all circuits, one finds a basis for the circuit vector space (hereinafter called a set of basic circuits). Let μ be the dimension of the space (μ is also called the nullity of the graph). All the $2^{\mu}-\mu-1$ circs (excluding the basic circuits themselves, and the null element) are computed. Since not every one of these circs is a circuit, a test is necessarily made to determine if the circ is a circuit.

In fact, we will prove that there are only four graphs whose circuit vector space consists of just circuits. It is useful to have the following definition.

Definition 1: A graph G is said to be reduced if

1. G is simple
2. G has no vertex of degree 0 or 1 and
3. if there is any vertex v in G of degree 2, then the two vertices adjacent to v are adjacent.

A ring-sum of a 1-circuit and any other circuit is always an edge-disjoint union of circuits. Consider the set P of all $\binom{p}{2}$ 2-circuits formed out of a set of p parallel edges. The ring-sum of any subset S of these circuits is either a circuit in P or an edge-disjoint union of circuits in P. No circuit passes through a vertex of degree 0 or 1. If a circuit contains an edge e_1 incident with a vertex v of degree 2, it must also contain the other edge e_2 incident with v. These considerations motivated the above definition.

Observe the following (see Figure 1):

1. If a reduced graph is not K_3, the complete graph on three vertices, then the nullity μ of the graph is at least two.
2. There is no reduced graph of one or two vertices. There is only one reduced graph of three vertices, viz., K_3. And only two reduced graphs are possible with four vertices namely K_4 and K_4-x. In all these graphs K_3, K_4 and K_4-x every circ is a circuit.
3. A reduced graph with five vertices and at most seven edges must have an edge-disjoint union of circuits.

LEMMA: Let G be a reduced graph with the number of vertices, $n \geq 6$. If G has no edge-disjoint union of circuits, then the number of edges $m \geq n + 3$. The converse is not true.

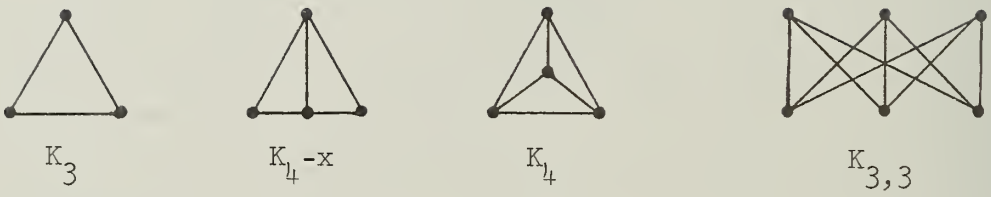


Figure 1: Graphs all of whose circs are circuits

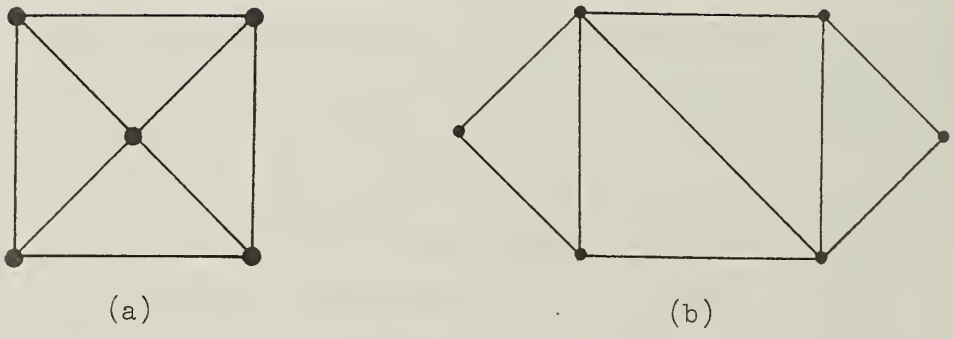


Figure 2: Graphs with m = n+3 and yet having edge-disjoint unions of circuits

PROOF: Since G is reduced, and has no edge-disjoint union of circuits, G must be connected. No vertex in G can be of degree less than 3; for otherwise G is either not reduced or has an edge-disjoint union of circuits. Therefore $e \geq \frac{3n}{2} = n + \frac{n}{2} \geq n + 3$, since $n \geq 6$.

The graphs in Figure 2 have $m = n+3$ and yet have edge-disjoint unions of circuits.

THEOREM 1: Only the four reduced graphs K_3, K_4, K_4-x and $K_{3,3}$ have all circs as circuits.

PROOF: The theorem is true for graphs with at most five vertices and seven edges by the observations made earlier. It is true for graphs with n vertices $n \geq 6$, and m edges, $m \leq n + 2$ by the lemma. For graphs with $n \geq 5$, and $m \geq n + 3$, the theorem will be proved by contradiction; it will be assumed that there exists a graph G which has $n \geq 5$, and is not $K_{3,3}$, and yet has all circs as circuits and draw a contradiction. If G is disconnected, we are done. Therefore, G is connected, and is either planar or nonplanar.

Case 1: G planar, $n \geq 5$, $m \geq n + 3$, that is nullity $\mu = m-n+1$ is at least four.

There are at least μ finite regions, and an infinite region defined on a plane representation of the graph G (see Harary (1969)). If any two finite regions are not adjacent, the circuits defining these regions are edge-disjoint. Likewise, every finite region must also be adjacent to the infinite region; otherwise the circuit defining the finite region is disjoint with the circuit defining the infinite region. The dual of G is therefore a complete graph $K_{\mu+1}$ of $\mu+1$ vertices, and $\mu+1 \geq 5$ which is

impossible. Thus, there must exist a pair of regions which are not adjacent, and hence an edge-disjoint union of circuits.

Case 2: G is nonplanar, and therefore G either has a proper subgraph homeomorphic to $K_{3,3}$, or has a subgraph homeomorphic to K_5.

Case 2.1: Let G have a subgraph homeomorphic to K_5. An edge-disjoint union of circuits of K_5 shown in Figure 3 is homeomorphic to a subgraph of G.

Case 2.2: G has a proper subgraph g homeomorphic to $K_{3,3}$. Since g is a proper subgraph, there exists an additional path between two vertices u, and v. If u and v correspond to vertices in the same independent set of vertices of $K_{3,3}$, there exists an edge-disjoint union of circuits (see Figure 4(a)). If u and v correspond to vertex u' in one independent set of vertices of $K_{3,3}$ and to v' in the other independent set, respectively, an edge-disjoint union of circuits exists (see Figure 4(b)).

Observe that the Theorem 1 does not give us information as to how many edge-disjoint unions of circuits a given graph can have. In general, a large fraction of the circs are edge-disjoint unions of circuits. The ratio of circuits to all vectors in the vector space for numerous classes of graphs tends to zero. Three such classes are shown in Figure 5. For wheel graphs, the nullity μ is n-1, the number of circuits is $c = \mu(\mu-1)+1$. For the modified ladder graphs, $\mu = \frac{1}{2}(n+2)$ and $c = \frac{\mu^2}{2} + \frac{5\mu}{2} - 3$. For complete graphs

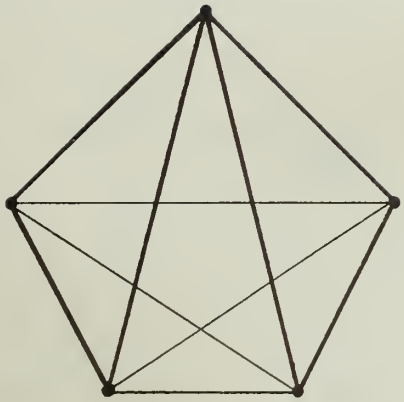
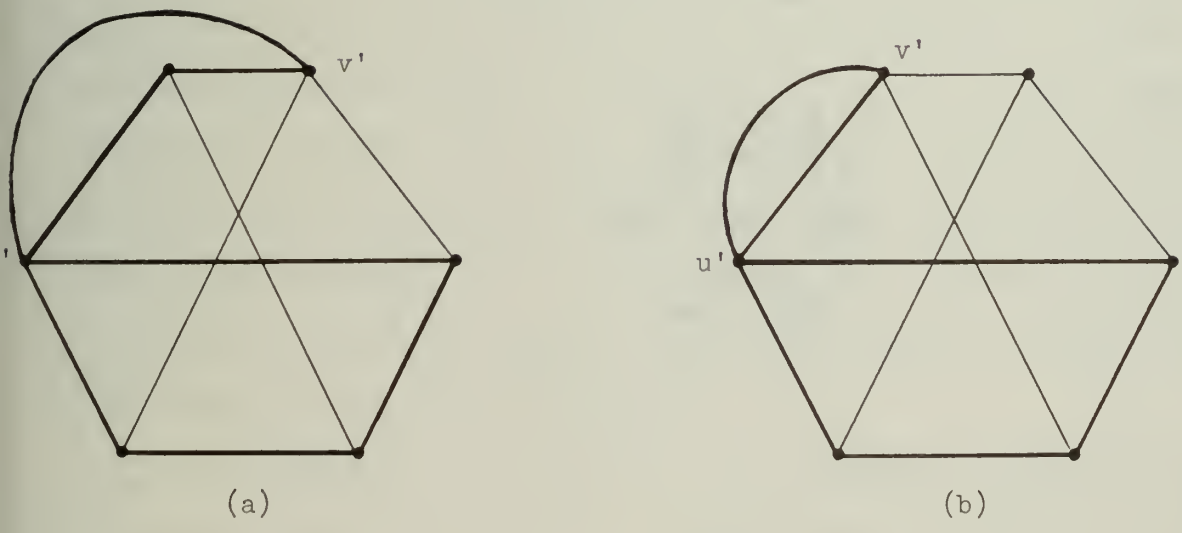
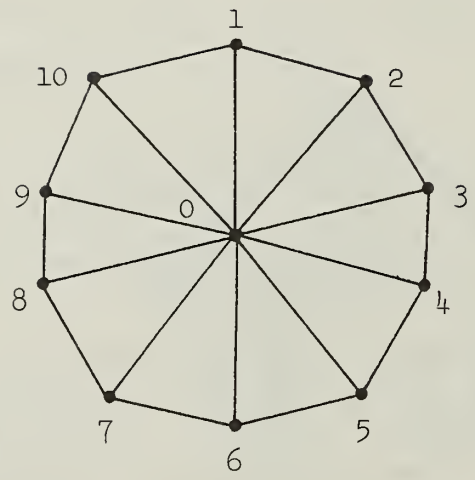


Figure 3: An Edge-disjoint Union of Circuits in a Complete Graph of Five Vertices

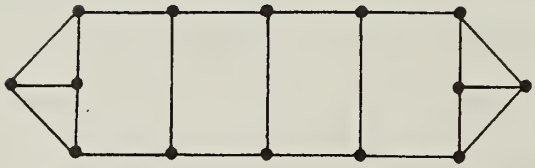


(a) (b)

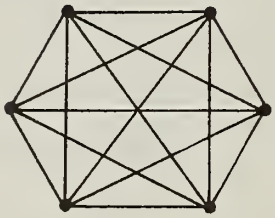
Figure 4: Edge-disjoint Unions of Circuits in a Nonplanar Graph Having a Proper Subgraph Homeomorphic to $K_{3,3}$.



(a) A Wheel Graph



(b) A Modified Ladder Graph



(c) A Complete Graph

Figure 5

$\mu = \frac{n^2}{2} - \frac{3n}{2} + 1$, and $c = \frac{1}{2} \sum_{s=3}^{n} \binom{n}{s} (s-1)!$ In all the three cases $\lim_{n\to\infty} \frac{c}{2^{\mu}} \to 0$. Following these assertions we make the following conjecture:

In any large reduced graph the ratio of circuits to the circs goes asymptotically to zero with the number of vertices, n.

To give a numerical example consider K_{10}, the complete graph on 10 vertices. The number of circuits is 556 014 and the number of edge-disjoint unions of circuits is 34 359 192 353. Thus the efficiency of an algorithm for finding all circuits would improve dramatically if the number of edge-disjoint unions of circuits it computes can be reduced. An attempt to compute only a subset of all circs, and yet find all circuits was made by Welch (1966). He chose the basis as a set of fundamental circuits, and tried to give an ordering of the fundamental circuits, namely that if there exist three fundamental circuits C_i, C_j and C_k, $i < j$, and $C_k \cap C_i \neq \emptyset$, and $C_k \cap C_j \neq \emptyset$, but $C_i \cap C_j = \emptyset$, then $k < j$. Such an ordering, if existed, would have reduced the number of edge-disjoint unions of circuits computed. Three years later Gibbs (1969) gave a counterexample and proved that there are graphs for which such an ordering does not exist. The corrected Welch-Gibbs algorithm computes all 2^{μ} linear combinations. We will shortly propose a method which reduces the number of edge-disjoint unions of circuits computed in the process of finding all circuits.

The dicircuits of a digraph do not constitute a vector space under any operation (see Williams (1972)). Thus this approach cannot be used with digraphs.

2.2 Search Algorithms

There are two kinds of search algorithms. Algorithms due to Char (1968) and Chan and Chang (1969) produce all permutations of i vertices chosen out of n vertices of a graph ($1 \leq i \leq n$). Any graph G of n vertices can be considered on a subgraph of the complete graph K_n. A circuit of G is also a circuit of K_n. Thus each circuit of K_n is tested to see if it contains an edge not present in G.

The second kind of search algorithms construct a dipath between vertices v_i and v_j and test to see if there is an edge e such that $f(e) = (v_j, v_i)$. We shall describe both kinds of algorithms in detail in Section 3.

2.3 Powers of Adjacency Matrix

Let A be the variable adjacency matrix* of a digraph. Then a non-null element A^p_{ij}, $p \geq 1$ is the set of all p-sequences from vertex v_i to vertex v_j. Since no dicircuit can be of length greater than n, we need compute only up to A^n. Observe that an edge-sequence need neither be a dipath nor a dicircuit. The crux of the problem is to avoid such (non-simple) edge-sequences. It is in this avoidance that different algorithms using the method

---

*Def. F1. A variable adjacency matrix A of a digraph $D = \langle V, E, f \rangle$ is a $n \times n$ symbolic matrix in which the (i,j)-element $A_{ij} = \sum_{\text{all } e_k} e_k$, where $e_k \in E$ such that $f(e_k) = (v_i, v_j)$. If there is no such e_k, then $A_{ij} = 0$. The powers of A, $p \geq 1$, are defined as follows: $A^1 \equiv A$ and for $p > 1$, $A^p \equiv A^{p-1} \cdot A = A \cdot A^{p-1}$ employing the usual symbolic multiplication.

vary. Kamae (1967) eliminates dicircuits and flowers* of length p from A^p and proceeds to the next power of A. Others, such as Danielson (1968), Ponstein (1966) do not delete the circuits and flowers, but determine whether or not a given edge-sequence is a dicircuit.

2.4 Edge Digraph of a Digraph

Consider a digraph $\mathcal{E}(D) = \langle J, Q, \psi \rangle$ derived from the given digraph $D = \langle V, J, f \rangle$ as follows: The vertex-set of $\mathcal{E}(D)$ is the edge-set of D, and the edge-set Q of $\mathcal{E}(D)$ contains an edge q, $\psi(q) = (e_1, e_2)$, e_1 and e_2 in J iff for some v, v_1 and v_2 in V, $f(e_1) = (v_1, v)$ and $f(e_2) = (v, v_2)$. Such a graph is called the edge digraph of D. (See Harary (1969))

Now consider a p-sequence S of a digraph D'. The p-sequence S yields a (p-1)-sequence in $\mathcal{E}(D')$, a (p-2)-sequence in $\mathcal{E}(\mathcal{E}(D'))$, and in general, a (p-k)-sequence in $\mathcal{E}^k(D') = (\mathcal{E}^{k-1}(D'))$, for $k \leq p$. Suppose now that all 1-dicircuits are reported and deleted from D', resulting in a subdigraph D. Every edge in $\mathcal{E}(D)$ will then correspond to a simple 2-sequence, and vice versa. An edge e of $\mathcal{E}(D)$ will correspond to a rooted 2-dicircuit** if it is an edge of a 2-dicircuit of $\mathcal{E}(D)$. Report and delete all 2-dicircuits of $\mathcal{E}(D)$, denoting the resulting digraph as $\mathcal{E}_2^1(D)$. ($\mathcal{E}_j^i(D)$ is the edge digraph of $\mathcal{E}_{j-1}^{i-1}(D)$ with all j-dicircuits deleted from it; $\mathcal{E}_1^0(D) \equiv D$.)

---

*Def. F2. A flower of a digraph is a subdigraph which is a union of a dipath and a dicircuit of length at least two such that the last vertex of the dipath is in the dicircuit, and this vertex is the only vertex in the dicircuit and dipath.

**Def. F3. A rooted k-dicircuit is a dicircuit of length k, with one of its k vertices specified as the root. Every k-dicircuit has k rooted k-dicircuits.

Assume inductively that there is a one-to-one correspondence between the simple p-sequences of D and the edges of $\mathcal{E}_{p-1}^{p-1}(D)$. Report and delete all its p-dicircuits, and denote the resulting digraph as $\mathcal{E}_p^{p-1}(D)$. Then each edge in $\mathcal{E}_p^{p-1}(D)$ corresponds to a p-dipath of D. Let $\mathcal{E}_p^p(D) = \mathcal{E}(\mathcal{E}_p^{p-1}(D))$. Now, each edge in $\mathcal{E}_p^p(D)$ corresponds to two adjacent edges of $\mathcal{E}_p^{p-1}(D)$, or what is the same, corresponds to two p-dipaths P_1 and P_2 of D having (p-1) edges in common. These two dipaths, therefore, form a simple (p+1)-sequence of D. Obviously, if the first vertex of P_1 (or P_2) is the same as the last vertex of P_2 (or P_1), their union gives a (p+1)-dicircuit D_1; and D_1 yields (p+1) rooted (p+1)-dicircuits, each of which has a corresponding edge in a (p+1)-dicircuit of $\mathcal{E}_p^p(D)$. If the edge e of $\mathcal{E}_p^p(D)$ does not lie in any (p+1)-dicircuit, then e corresponds to a (p+1)-dipath of D.

Thus, given a digraph D, we report all 1-dicircuits, delete them and proceed to take the edge digraph of D; report and delete all 2-dicircuits of the edge digraph of D, and so on. Observe that a p-dicircuit will be reported p times if an algorithm is implemented using the method as described.

Computationally it will be advantageous to perform some initial simplification on the given graph before actually proceeding to find all circuits. Parallel edges, self-loops, vertices of degree 0 (in-degree or out-degree 0, for digraphs) vertices of degree 2 (for undirected graphs), vertices of in-degree and out-degree 1 can all be taken care of initially. Then if the given graph G is undirected, decompose it into blocks. If G is a digraph decompose it into fragments. The circuit finding algorithms can now be applied on these subgraphs. This is advantageous because such editing algorithms have a time bound of $O(\max(m,n))$ (see Hopcroft and Tarjan (1973)), whereas the circuit finding algorithm is essentially an exponential algorithm and does not fall in any of the classes defined by Karp (1972) and Cook (1971).

3. Analyses of Algorithms

Before we proceed to derive time bound T_{ALG}'s for the algorithms, let us keep in mind the caveat that as far as possible an analysis of the 'algorithm' and not an analysis of an 'implementation' of the algorithm is attempted. However, an analysis of an algorithm without any regard to an implementation is not always possible. So much so, whenever a particular implementation is known to improve the speed of an algorithm, we assume that implementation and analyse. We assume that all the data references made by an algorithm are to data elements stored in a random access memory and that once a circuit is found it is output to an external storage medium (and also stored simultaneously in the random access memory if the algorithm requires at a later stage circuits computed earlier).

3.1 Circuit Vector Space Algorithms

Welch (1966)-Gibbs (1969) algorithm: The Welch-Gibbs algorithm chooses a set of fundamental circuits as a basis for the circuit vector space. The algorithm as given by Gibbs (1969) would be reporting some edge-disjoint unions of circuits as circuits and possibly missing some circuits if a set of non-fundamental but basic circuits were used as a starting set.

Let $B = \{B_1, B_2, \ldots, B_\mu\}$ be the starting set of fundamental circuits. We build up iteratively two sets S and Q such that when the process terminates S contains all circuits and Q contains all circs. Initially both S and Q contain fundamental circuit B_1 only. At the beginning of the i^{th} iteration $2 \leq i \leq \mu$, S and Q contain all circuits and circs obtained so far. During the i^{th} iteration we obtain circuits and circs by taking ring-sum of B_i with each

element M of set Q, and then add them to sets S and Q, respectively. If M and B_i do not intersect, their ring-sum R will be an edge-disjoint union of circuits, and vector R is included in a set Q_t which contains edge-disjoint unions of circuits computed during the i^{th} iteration. If M and B_i do intersect, their ring-sum R could be a circuit or an edge-disjoint union of circuits, and then vector R is included in a set S_t which contains the ring-sums of a circ M and the fundamental circuit B_i if M intersects with B_i. Thus, circuits computed during the i^{th} iteration are elements of the set S_t. It can be shown that a vector W in S_t corresponds to a circuit iff it does not contain or is contained in another vector in S_t. Vector W is chosen from S_t, and compared with every element in S_t. If for any vector R' in S_t, $W \subseteq R'$ then R' is not a circuit, and the vector R' is deleted from S_t and added to Q_t. If, on the other hand, $W \supseteq R'$ then W is not a circuit, and the vector W is deleted from S_t and added to Q_t. Choose a new vector from S_t as W and repeat the process. At the end of this search for containment, S_t will contain all and only circuits computed during the i^{th} iteration. The set S_t and the fundamental circuit B_i are added to both S and Q, set Q_t is also added to Q, and we proceed to the next iteration.

Observe that the set of circs Q contains exactly 2^i-1 vectors at the end of the i^{th} iteration. Considering a worst case when every vector in Q will be intersecting with the fundamental circuit-vector B_{i+1}, the set S_t of all ring-sums of a vector M in Q and B_{i+1} such that M and Q intersect, will contain at most 2^i-1 vectors. Let N_i be the number of comparisons made in the testing for the minimality, with respect to containment of the vectors in S_t, during the i^{th} iteration. Then,

$$N_i = \alpha_i |S_t|^2 \quad \text{where} \quad 0 < \alpha_i < 1$$

If S_t contained $\beta_i(2^i-1)$ vectors, $0 < \beta_i < 1$, then

$$N_i = \alpha_i \beta_i^2 (2^i-1)^2$$

Numbers α_i and β_i depend on the structure of the graph. The extreme values are not likely to be true for any graph. And the total number of comparisons made is $\sum_{i=2}^{\mu} \alpha_i \beta_i^2 (2^i-1)^2 = O(2^{2\mu})$. Thus, the time bound

$$T_{W\text{-}G} = O(2^{2\mu}). \qquad (3.1.1)$$

All the 2^{μ}-1 vectors computed have to be stored, and the storage required is of the order of $m \cdot 2^{\mu}$ bits.

Algorithm of Rao and Murti (1969): The R & M algorithm differs from W-G algorithm in that that a vector which is a ring-sum of some basic circuit-vectors is not tested for its minimality with respect to containment of another vector. Instead an attempt is made to construct a spanning tree of the given graph G using all but one edges present in a circ. We will succeed in this attempt iff the circ is a circuit. Observe that it is not necessary to have in the memory all the circuits and/or edge-disjoint unions of circuits computed so far, and that every circ will have to be computed as a ring-sum of some basic circuits. If a set of fundamental circuits is chosen as the basis, and if the chords are labeled from 1 to μ, then we need only store the characteristic submatrix B_{12} of the fundamental circuit matrix B_f (see Deo (1974)). The matrix B_{12} is a binary matrix of size $\mu \times (n-1)$.

Let R be the ring-sum of k fundamental circuits, and let it contain ℓ edges. Without loss of generality, $R = \{c_1, c_2, \ldots, c_k, b_1, b_2, \ldots, b_{\ell-k}\}$ where c_i's stand for the chords and b_i's for tree branches with respect to a spanning tree T. If R were a circuit, a spanning tree containing any $(\ell-1)$ edges of R, and in particular a spanning tree containing all the tree branches of R and the (k-1) chords c_i for $2 \leq i \leq k$, must exist. On the other hand, if R is an edge-disjoint union of circuits, it properly contains a circuit which could be made a fundamental circuit with respect to some spanning tree. Thus,

the chord c_k is added to the spanning tree T, and a search is made to see if there exists a tree branch of T not in R that could be deleted from T to break the fundamental circuit containing the chord c_k. If there is no such branch b available, obviously R properly contains this circuit, and R is not a circuit. Otherwise, the chord c_k is introduced into T and b is deleted from T. Call the resulting spanning tree as T. The set R now contains one less a chord than it did prior to the transformation. This process is repeated for all chords c_i, $2 \leq i \leq k$, if necessary. If all these chords could be made as tree branches, then R is a circuit, otherwise it is not.

All $2^{\mu}-1$ vectors are computed and tested to see if a vector corresponds to a circuit. A vector is computed as a ring-sum of some basis vectors. The number of ring-sum operations performed is given by

$$\sum_{k=2}^{\mu} k\cdot\binom{\mu}{k} = \mu \cdot 2^{\mu-1} - \mu$$

Given a vector R, a ring-sum of k basis vectors, testing if it corresponds to a circuit, in the worst case, needs the introduction of all k-1 chords. The total number of chords used is, again, $\mu \cdot 2^{\mu-1} - \mu$. Thus for the Rao and Murti algorithm we have the upper bound

$$T_{R\&M} = O(\mu \cdot 2^{\mu}) \qquad (3.1.2)$$

It is easy to see that the lower bound is also

$$O(\mu \cdot 2^{\mu}) = T_{R\&M} .$$

Algorithm of Maxwell and Reed (1965): All the $2^{\mu}-1$ circs are computed first; then each circ R is compared with every other circ to see if R is a circuit. A worst case analysis would immediately give an upper bound of $O(2^{2\mu})$ which is also an upper bound for Welch-Gibbs algorithm. However, direct

comparison shows that for any graph G, Welch-Gibbs algorithm makes lesser number of comparisons for containment than Maxwell and Reed algorithm.

Thus, of the three circuit vector space algorithms described Rao and Murti algorithm is the fastest, and requires the least storage. (However, see Section 4.)

3.2 Search Algorithms

These algorithms are, in general, applicable to both undirected, as well as directed graphs unlike the circuit vector space algorithms which are applicable only to undirected graphs.

Char's Algorithm (1968): The algorithm has a list of all circuits, a master circuit matrix, of a complete graph K_p. If all circuits of a graph G having n vertices, $n \leq p$, are required, each circuit of K_p of length at most n is tested to see if it contains an edge not in G. If all the edges in the circuit considered are also edges of G, the circuit is reported. Assuming that the circuits of a complete graph K_j, $j \leq n$ are listed lexicographically earlier than the circuits of the complete graph K_{j+1}, the time bound is

$$T_C = O\left(\frac{1}{2} \sum_{s=3}^{n} \binom{n}{s} \cdot (s-1)!\right) \qquad (3.2.1)$$

Storage is required for all the circuits of the complete graph, K_n. Assuming each circuit is stored in one word, the total requirement is $\sum_{s=3}^{n} \binom{n}{s} (s-1)!$ words. Clearly, storage-wise Rao and Murti algorithm is vastly superior to Char's algorithm. Since no closed form expression is available for Eq. (3.2.1), it is difficult to compare Char's algorithm with Rao and Murti algorithm. Assuming that the sum on the right-hand side of Eq. (3.2.1) is

proportional to n^n, Rao and Murti algorithm would be superior to Char's when

$$\mu \cdot 2^{\mu} \leq n^n,$$

i.e., $$\mu + \log_2 \mu \leq n \log_2 n.$$

That is, Rao and Murti algorithm is superior to Char's, when the nullity μ is linear in the number of vertices, or equivalently, if the number of edges m is linear in n.

Algorithm of Chan and Chang (1969): The Chan and Chang algorithm uses a modified variable adjacency matrix. Actually, it generates all permutations and tests if there exists a directed edge between two consecutive vertices in the permutation and between a chosen vertex and the first vertex. Given an initial sequence 1_2_3_..._n_1, the permutation P is pivoted at the r-th element p_r in P, $1 < r \leq n$. All integers p_i, $i \geq r$, are successively incremented by 1. The number n when incremented results in r, by definition. Once a new permutation is thus obtained every subsequence 1_2_3_.._s is tested to see if a dicircuit exists with 1_2_3_.._s_1 as its vertex-sequence. We generate all permutations containing 1 and then delete 1 and keep 2 as the first vertex in the next set of permutations. In general, once all permutations with k as the first vertex are generated, delete k and take k+1 as the first vertex and proceed. Terminate if k=n. The time bound for this algorithm is

$$T_{C\&C} = O\left(\sum_{i=3}^{n-1} i!\right) \qquad (3.2.2)$$

Algorithm of Roberts and Flores (1966), Floyd (1967) and Tiernan (1970): The algorithm which was originally given by Roberts and Flores (1966), for finding all Hamiltonian dicircuits of a digraph, in actuality generated all dicircuits. In Floyd (1967), it appears as a nondeterministic algorithm.

Tiernan (1970) gives a deterministic version of the algorithm, and proves its finiteness and correctness. The algorithm as given in Roberts and Flores (1966) and Tiernan (1970) uses the successor listing. Every vertex v of the digraph has a row in a matrix, and all the end-vertices of edges incident out of the vertex v are listed as the successors of v. But improvement in the speed occurs if the adjacency matrix of the digraph is used.

The algorithm constructs a dipath P and tests if there is a directed edge from the last vertex to the first vertex of the dipath P. If there is such a directed edge a dicircuit is found, and is duly reported. The construction of a dipath is as follows: Let p_1 be the first vertex of the dipath being constructed. If there is a directed edge (p_1, p_2) such that $p_2 > p_1$, add the edge to the dipath, and the length of the dipath increases by 1. Suppose that a k-dipath P is constructed, so far. Let its vertex-sequence be $P = p_1\text{-}p_2\text{-}\ldots\text{-}p_{k-1}\text{-}p_k$. An attempt is now made to extend the dipath P. This extension is possible only if there is an edge (p_k, v) such that v is not a vertex in P, because otherwise a flower blooms! The extension is made only if v is not 'forbidden' and $v > p_1$. This device ensures that every dicircuit is reported once only by defining a one-to-one and onto mapping of dicircuits to rooted dicircuits, rooted at its lowest numbered vertex. If no directed edge (p_k, v) satisfying the above conditions exists, a check is made to see if a dicircuit can be found as described earlier. If a circuit is confirmed, it is reported. In any case, the last vertex of the dipath p_k is deleted. Call the subdipath $p_1\text{-}p_2\text{-}\ldots\text{-}p_{k-1}$ as P, and the above extension process is now applicable to the new dipath P. The last vertex p_k is not available for extension, because otherwise a dipath already reported would result. Thus, the vertex p_k is made 'forbidden' from the vertex p_{k-1}. This does not necessarily mean

that p_k is forbidden from other vertices, or that p_k will be forbidden forever from p_{k-1}. Once the last vertex (which is p_{k-1}) of the new dipath P is similarly deleted, the vertex p_k is made available from p_{k-1}.

The algorithm starts with $p_1 = 1$. A new first vertex $p_1 \leftarrow p_1 + 1$ is used after the current first vertex p_1 is deleted. The algorithm terminates after detecting that the vertex $p_1 = n$.

Tiernan (1970) claims that his algorithm is the "theoretically most efficient" search algorithm. The claim has to be disputed for two reasons:

1. Tiernan's algorithm does consider a dicircuit more than once during the search.

2. Even a search algorithm which considers each dicircuit once and only once cannot be said to be theoretically most efficient.

To show that Tiernan's algorithm does consider a dicircuit more than once, we exhibit the graph in Figure 6. The dicircuit 1-2-3-1 will be found earlier,

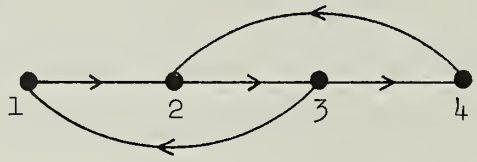


Figure 6: A Graph for the Tiernan's Algorithm

but will not be reported until the dipath 1-2-3-4 is explored. Also, in testing for a directed edge from vertex 4 to vertex 1, a directed edge (4, 2) is found giving the dicircuit 2-3-4-2; yet, this dicircuit will not be reported until the first vertex of the dipath becomes 2. The above observations are implementation independent, i.e., whether a successor listing is used or a binary adjacency matrix is used these dicircuits are found once before they are found again and reported.

Secondly, in trying to consider a dicircuit once only, Tiernan's algorithm considers each directed edge a number of times. Minimizing the number of examinations of each individual edge also improves the efficiency. Thus, a more efficient algorithm might be possible by minimizing the number of considerations of each dicircuit in conjunction with a minimization of the number of examinations of each edge.

To estimate a lower bound for the RFFT algorithm, observe that for each dipath considered, a test will be made to see if a directed edge exists from the last vertex of the dipath to the first vertex, and the list containing the forbidden vertices from the last-but-one vertex of the dipath is cleared. The list is implemented as a binary, linear array; n operations are required to clear the list. Now, the algorithm considers all dipaths and only dipaths with the property (which for short we shall call property π), that the first vertex is the least numbered vertex in the dipath. Thus, the time bound is

$$O(n \cdot (\text{number of dipaths with property } \pi)) = T_{RFFT} \qquad (3.2.3)$$

Since for each k-dicircuit there are at least (k-1) dipaths with the property π, the algorithm is <u>not</u> linear with respect to the number of dicircuits, as claimed by Tiernan (1970).

Weinblatt's Algorithm (1972): Weinblatt suggests an algorithm similar to the RFFT algorithm. A dipath P is built up, and if there exists an edge from the last vertex of the dipath P to another vertex in P, a dicircuit thus formed is immediately reported, in contrast to the RFFT algorithm. Also, the Weinblatt's algorithm considers each directed edge once only, and makes use of parts of dicircuits reported earlier to construct new dicircuits. For example, if a dipath P has its last vertex on a dicircuit D_1, a test is made

to see if any other vertex of P is on the dicircuit D_1. Or, if the dicircuit D_1 had a vertex in common with another dicircuit D_2 which in turn had a vertex on the dipath P, these two dicircuits D_1 and D_2 are used to give a new dicircuit. In general, D_1, D_2, ..., D_k may be used similarly to give a new dicircuit. However, this necessitates a test to see if a part of the new dicircuit as given by D_1, D_2, ..., D_k is also not given by another set of dicircuits D_{i_1}, D_{i_2}, ..., D_{i_j}.

The algorithm is recursive and giving a non-trivial bound on the depth of recursion is a difficult combinatorial problem, and the analysis of the algorithm is thereby complicated. We will not therefore attempt to analyze the algorithm. However, it appears to be faster than the RFFT algorithm.

3.3 Algorithms Using Powers of Adjacency Matrix

The algorithms will be described first, and then all of them will be directly compared at the end of this subsection. (Note that the Def. F2 for A^p, powers of variable adjacency matrix assumes that no entries in A^{p-1} are modified as will be done in the following algorithms. Hence, in these algorithms, $A^{p-1} \cdot A \neq A \cdot A^{p-1}$, in general.)

Yau's Algorithm (1967): A nonzero entry $A^p ij$ gives all the p-sequences from vertex v_i to vertex v_j as a sum of products, each product giving a p-sequence. If a p-sequence is non-simple, then it must contain a k-dicircuit, for $k < p$. Yau's algorithm computes A^p and tests each p-sequence in a non-zero entry of A^p to see if it contains any k-dicircuit, $k < p$, reported earlier. If it does, that p-sequence is deleted from the sum. Obviously, if a

p-sequence does not contain any k-dicircuit, $k < p$, it must be simple. If the p-sequence is from a non-diagonal entry, it is a dipath; otherwise it is a dicircuit and hence is reported. The diagonal entries in A^p are set to zero once all the p-dicircuits, if any, are reported, and p is incremented by 1 and $A^p \leftarrow A^{p-1} \cdot A$, and the process is repeated. The algorithm terminates if either $p = n$ or if for some p, A^p is an all-zero matrix. This termination clause is common to all algorithms described in this subsection, and we shall not repeat it.

Ponstein's Algorithm (1966): If A^p contains all simple p-dipaths and only those, then we can construct A^{p+1} satisfying the same property as follows: Compute $A \cdot A^p$ and $A^p \cdot A$. Include in A^{p+1}_{ij} only those (p+1)-sequences which are present in $(A \cdot A^p)_{ij}$ and $(A^p \cdot A)_{ij}$. Assume that a (p+1)-sequence R in $(A \cdot A^p)_{ij}$ is non-simple. Since A^p contained only simple edge-sequences, R is non-simple because of an edge (v_i, v_k), where v_i was already an interior vertex on the dipath from v_k to v_j. Observe that in the (p+1)-sequence R, the only vertex of out-degree 2 is v_i. No (p+1)-sequence in $(A^p \cdot A)_{ij}$ can be an edge-sequence with v_i having an out-degree of 2, because this can happen only if A^p_{ii} were non-null. Thus R will not appear in $A^p \cdot A$. It is easy to see that a simple p-sequence S will appear in both $A \cdot A^p$ and $A^p \cdot A$. The diagonal entries give rooted (p+1)-dicircuits. Report and delete these dicircuits. Obviously, A^{p+1} now satisfies the stated property.

Danielson's Algorithm (1968): A major source of inefficiency in Yau's algorithm is in the testing to see if an edge-sequence is simple. Danielson's algorithm employs a much simpler test.

All edges incident into a vertex v are labeled with v. Using these labels construct the variable adjacency matrix A. Construct another matrix A^0, obtained from A by replacing every nonzero entry by the null string λ. Any

string s concatenated (multiplied) with λ gives the same string s. Call as A^2 the product $A \cdot A^0$, and $A^p = A \cdot A^{p-1}$, $p \geq 3$. Then an edge-sequence appearing in A^p also gives corresponding vertex-sequence but with its end-vertices missing. Assuming that A^{p-1} contains only (p-1)-dipaths, a p-sequence from vertex v_i to vertex v_j is non-simple iff v_i appears in the p-sequence. All non-simple edge-sequences are deleted from A^p, and all p-dicircuits are reported and deleted. Obviously, A^p now contains only p-dipaths. Observe that because of this peculiar edge-labeling, only premultiplying of A^{p-1} with A is allowed.

The above three algorithms report the rooted dicircuits. If a dicircuit need be reported only once, we may report the unique rooted dicircuit of every dicircuit rooted at the smallest numbered vertex in the dicircuit.

Kamae's Algorithm (1967): This algorithm uses the binary adjacency matrix, X. Powers of this matrix are computed in real field; thus, a non-zero entry X_{ij}^{p-1} gives the <u>number</u> of edge-sequences from vertex v_i to vertex v_j. Assume, as before, that X^{p-1} gives the number of (p-1)-dipaths only. Then to compute X^p giving the number of p-dipaths and to report all p-dicircuits during such computation, it will be necessary to find the number of non-simple edge-sequences. If the matrix X^p were to be obtained by postmultiplying X^{p-1} by X, all non-simple edge-sequences will be flowers (see Def. F2.). Therefore, it will be necessary to compute the number of p-flowers from vertex v_i to vertex v_j and then subtract it from $(X^{p-1} \cdot X)_{ij}$. This number is the sum of the number of flowers with k-dicircuits (containing the vertex v_j) and with (p-k)-dipaths from vertex v_i to vertex v_j, $2 \leq k \leq p-1$. Every k-dicircuit containing v_j but not v_i is considered, and the number of (p-k)-dipaths from v_i to v_j is computed. Now, the length of the dicircuit k takes values from 2

to (p-1), and this computation is performed for these values of k and the sum of the number of flowers found for each k gives the total number of flowers from vertex v_i to vertex v_j. Construct a flower matrix F, such that F_{ij} gives the number of flowers from vertex v_i to vertex v_j. Then

$$X^p \leftarrow X^{p-1} \cdot X - F$$

Now, X^p_{ij} gives the number of simple p-sequences from v_i to v_j. A non-zero diagonal entry X^p_{ii} gives the number of p-dicircuits containing the vertex v_i. Once these dicircuits are found and reported, set the diagonal entries to zero, increment p by 1 and repeat the procedure.

To find the p-dicircuits, compute a matrix Z defined as follows:

$$Z_{ij} = X^{p-1}_{ji} \cdot X_{ij}$$

The above equation self-explains that Z_{ij} will be nonzero iff there is a p-dicircuit containing the directed edge (v_i, v_j). For details of tracing the dicircuits using matrix Z, and for the details of computing the flower matrix F, the reader is referred to Kamae (1967).

The algorithms of Danielson (1968), Kamae (1967), Ponstein (1966), and Yau (1967) will now be directly compared with each other. As was mentioned earlier in Section 2, these algorithms differ only in identifying the non-simple sequences, and it is this identification that we compare.

Yau's algorithm identifies a non-simple p-edge-sequences by testing it to see if it contains any k-dicircuit, $k < p$, reported earlier. Thus, at most N_{p-1} tests are performed, where N_{p-1} is the number of dicircuits of length at most (p-1). Ponstein finds a non-simple p-edge-sequence of $(A^{p-1} \cdot A)_{ij}$ by testing to see if the sequence is absent in $(A \cdot A^{p-1})_{ij}$, and this would require at most s_p tests, where s_p is the number of p-edge-sequences

in $(A \cdot A^{p-1})_{ij}$. All flowers from the vertex v_i to vertex v_j are computed by Kamae's algorithm and this requires considering all k-dicircuits, $k < p$, containing v_i but not v_j. For each k-dicircuit considered, the edges of incident out of the vertices of the dicircuit are deleted, and dipaths in the resulting digraph connecting the vertex v_i with vertex v_j are to be counted. This would, in turn, require that the flowers of the resulting digraph be enumerated, and so on. Thus, Yau's algorithm is faster than Kamae's.

Danielson's algorithm determines whether or not a given p-sequence from vertex v_i to vertex v_j is simple by testing if the vertex-sequence contains the vertex v_i. Only one operation is required for each sequence. Since only premultiplying was allowed, any non-simple edge-sequence is a flower. The time bound is therefore

$$T_D = O(\text{Number of all dipaths} + \text{Number of all flowers}). \qquad (3.3.1)$$

For each sequence Ponstein's algorithm makes as many tests as there are terms in $(A \cdot A^p)_{ij}$, and Yau's algorithm makes, at most, as many tests as there are previously reported dicircuits. Danielson's algorithm is therefore superior to those of Yau, Ponstein or Kamae.

Comparing with RFFT algorithm, the set of dipaths with property π is a subset of all dipaths, and

$$T_D \geq T_{RFFT} . \qquad (3.3.2)$$

3.2.4 Edge Digraph Algorithm

The concept of edge digraph is used in an algorithm by Cartwright and Gleason (1966). The method was described in detail in Section 2.4 and we will not repeat it here.

The union of two simple p-dipaths P_1 and P_2 will be a (p+1)-dicircuit if the first vertex of P_1 is the same as the last vertex of P_2 and there is a common (p-1)-subdipath in P_1 and P_2. Thus, to find a (p+1)-dicircuit, a pair of p-dipaths P_1 and P_2 having a (p-1)-dipath in common, and also having a last vertex of P_2 as the first vertex of P_1 must be found. Thus, for each (p+1), $N_p \cdot N_p$ p-sequence pairs are considered. So a lower bound is

$$O\left(\sum_{p=1}^{n-1} N_p^2\right) = T_{C\&G} \qquad (3.4.1)$$

where N_p is the number of simple p-sequences.

The number of simple p-sequences is not less than the number of dipaths with the property π of the RFFT algorithm. Moreover Eq. (3.4.1) is quadratic in N_p.

The RFFT algorithm for dicircuits is, therefore, the fastest of all algorithms analyzed, and Rao and Murti algorithm is the best of the three circuit vector space algorithms analyzed. Weinblatt's algorithm appears to be faster than Tiernan's algorithm for the reason that parts of some circuits computed earlier, are used to produce new circuits.

4. Circuit-graph

As was already pointed out in Section 3.1 the main source of inefficiency in the algorithms using the circuit vector space approach is in the wasted computation of edge-disjoint unions of circuits. It would be ideal if we computed only the circuits. Toward this end we define a new graph -- a circuit-graph of a graph -- and demonstrate its use in the computation of all circuits of a graph.

Let $B = \{B_1, B_2, \ldots, B_\mu\}$ be a set of basic circuits. Observe the following:

1. Any circuit C of a graph G can be expressed as a ring-sum of b basic circuits, $1 \leq b \leq \mu$,

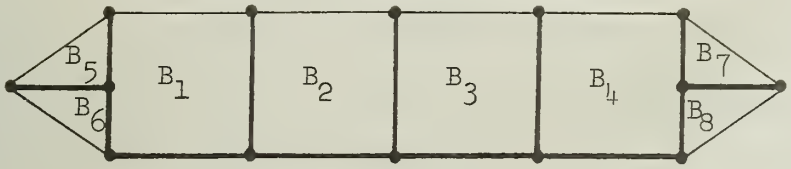
$$C = B_{i_1} \oplus B_{i_2} \oplus \ldots \oplus B_{i_b}.$$

Consider this set of basic circuits, $S = \{B_{i_1}, B_{i_2}, \ldots, B_{i_b}\}$. It is not possible to partition S into two nonempty blocks S_1 and S_2 such that the ring-sum C_1 of all circuits in S_1 is disjoint with the ring-sum C_2 of all circuits in S_2.

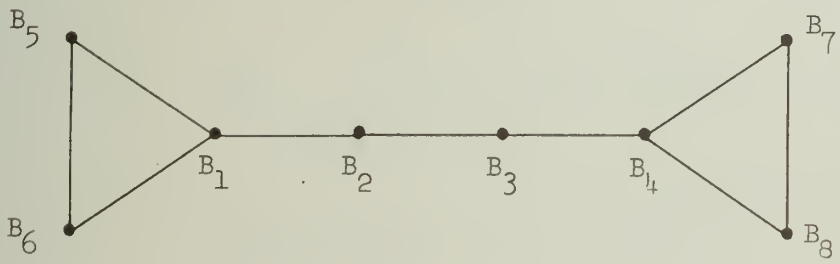
2. If two circs C_1 and C_2 do not intersect, i.e. if $C_1 \cap C_2 = \emptyset$ then their ring-sum $C_1 \oplus C_2$ cannot be a circuit. However, if C_1 and C_2 do intersect, $C_1 \oplus C_2$ may or may not be a circuit.

Definition 2: A circuit-graph of a graph $G = \langle V,E,f \rangle$ with respect to a set B of basic circuits of G is a graph $\mathcal{C}(G,B) = \langle V_C, E_C, f_C \rangle$. The vertex set $V_C = B$, and for an edge $e \in E_C$, $f_C(e) = \{B_i, B_j\}$ iff $B_i \neq B_j$ and $B_i \cap B_j \neq \emptyset$ for $1 \leq i,j \leq \mu$.

An example of a circuit-graph of G_L, the modified ladder graph is given in Figure 7. The set of fundamental circuits with respect to the spanning tree shown in bold lines is chosen as B. For a different set of basic circuits we would, in general, get a different circuit-graph. But the number of vertices of a circuit-graph of G is always μ, the nullity of G. Observe that if the graph G has β blocks, then every circuit-graph (with respect to any set of basic circuits) of G will have β components. Consider a subset $S = \{B_{i_1}, B_{i_2}, \ldots, B_{i_b}\}$ of vertices of a circuit-graph $\mathcal{C}(G,B)$ of G. If $R = B_{i_1} \oplus B_{i_2} \oplus \ldots \oplus B_{i_b}$ is a circuit of G, then there must be a path in $\mathcal{C}(G,B)$ between any two vertices B_{i_k}, and B_{i_ℓ} in S. Thus, with any circuit C



(a) G_L



(b) A Circuit-Graph G_C of G_L

Figure 7.

of G we can associate a connected subgraph of $\mathcal{C}(G,B)$ with vertices $B_{i_1}, B_{i_2}, \ldots, B_{i_b}$. Only those subsets of vertices of $\mathcal{C}(G,B)$ which are the vertex-sets of connected subgraphs of $\mathcal{C}(G,B)$ are of interest. Any representative of these connected subgraphs (with the same vertex-set) may be taken. A convenient subgraph is the connected induced subgraph.* This correspondence between circuits of G and connected induced subgraphs of $\mathcal{C}(G,B)$ is one-to-one but not necessarily onto, because of observation 2.

To compute all circuits of a graph G, then the procedure is as follows:

Procedure CCUBE: [Compute all Circuits using a Circuit-graph]:

1. Take a set B of basic circuits of G.
2. Construct the circuit-graph $\mathcal{C}(G,B)$.
3. Find the vertex-sets, V's of all connected, induced subgraphs of $\mathcal{C}(G,B)$.
4. Compute the ring-sum R of the basic circuits corresponding to the vertices in the vertex-set V.
5. Test if R is a circuit. If yes, report the circuit.
6. Take a new vertex-set V, and go to Step 4. If no new vertex-set V exists, stop.

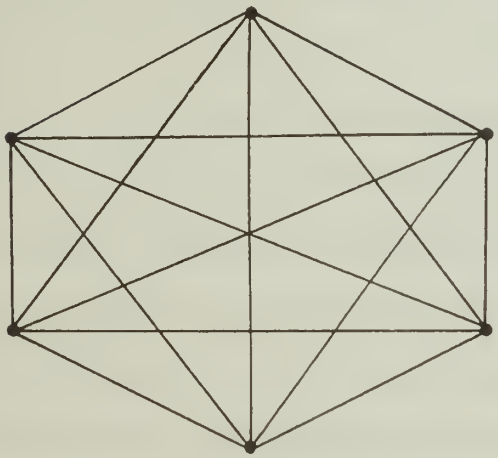
Observe that a test in Step 5 to decide if S is a circuit is necessary.

Let us apply this procedure to G_L (Figure 7). The circuit-graph G_C produced in Step 2 is given in Figure 7(b). (The basis B was the set of fundamental circuits with respect to the spanning tree shown in bold lines.)

---

*Def. F4: An induced subgraph $g = \langle V',E',f' \rangle$ of a graph $G = \langle V,E,f \rangle$ is a maximal subgraph of G with the vertex set V'.

(a) G_L with a Hamiltonian Path as a Spanning Tree



(b) Circuit-graph $\mathcal{C}(G_L, B')$
(B' is the set of fundamental circuits with respect to the spanning tree shown in (a))

Figure 8.

There are exactly $\frac{\mu^2}{2} + \frac{5\mu}{2} - 3$ connected induced subgraphs,* where μ is the number of vertices of G_C (nullity of G_L). That is every connected induced subgraph of G_C corresponds to a unique circuit of G_L, and there is no wasted computation. However, for a different basis this property may not hold. In fact, if a Hamiltonian path is chosen as the spanning tree to produce a set B' of fundamental circuits then $\mathcal{C}(G_L,B')$ is a complete graph of μ vertices (see Figure 8). This brings us to an important question:

> Q: What basis B of the circuit vector space of the graph G be chosen so that $\mathcal{C}(G,B)$ has a minimum number of connected induced subgraphs?

No precise answer has been found. Approximate answers to this question are: Choose that basis B for which

> A1: the sum of lengths of the basic circuits is a minimum
>
> A2: the number of edges of $\mathcal{C}(G,B)$ is a minimum.

The algorithm of Paton (1969) attempts to produce a set of fundamental circuits satisfying A1 provided the vertices are labeled 'properly', and no criterion is known to label these vertices 'properly'. (See also Prabhaker (1972).) However, it may be noted that Paton's algorithm never takes a Hamiltonian path (if it exists) of a graph G, as the spanning tree to produce the set of fundamental circuits, unless of course, G is a circuit. That is, if we use this algorithm to find a basis B, the circuit-graph $\mathcal{C}(G,B)$ may not be complete. But are there graphs for which every circuit-graph is complete? We have the following theorem which is a consequence of Theorem 1.

THEOREM 2: If a reduced graph G is not one of the four graphs K_3, K_4, K_4-x, or $K_{3,3}$ then G has a circuit-graph which is not complete.

---

*We are ignoring the trivial connected induced subgraphs which are the individual vertices of the graph.

A brute force algorithm for finding the sets of vertices of connected induced subgraphs of a graph can be easily given incorporating a connected components algorithm (say of Hopcroft and Tarjan (1973)) which will have a time bound of $O(\mu \cdot 2^{\mu})$. For each set of vertices of a connected induced subgraph of $\mathcal{C}(G,B)$, only at most μ ring-sum operations are required (Step 4). To test if a circ is a circuit again requires at most μ operations (using a test similar to that made by Rao and Murti algorithm). Now, assuming that an algorithm for finding the vertex-sets of all connected induced subgraphs of a graph has a time bound of $O(N)$, (where N is the number of connected induced subgraphs of $\mathcal{C}(G,B)$) the algorithm for all circuits of G based on CCUBE will have a time bound of $O(\mu \cdot N)$. Because of Theorem 2, we can always choose a circuit-graph which is not complete, and hence $N < 2^{\mu} - \mu - 1$. Then our algorithm is faster than that of Rao and Murti algorithm. However, no algorithm for finding a basis of circuit vector space of a graph G and answering the question Q is known to the authors. Thus the success of an algorithm based on CCUBE depends on our being able to give efficient algorithms for finding the vertex-sets of connected induced subgraphs of a graph, and for finding a 'good' basis for the circuit vector space of a graph.

5. Conclusions

Of all the analyzed algorithms for finding all dicircuits of a digraph, the RFFT search algorithm is the fastest, and requires least storage. Weinblatt (1972) algorithm is not analyzed because of its extreme complexity due to the recursion. It appears that Weinblatt's algorithm will be faster than RFFT algorithm. The time bound for RFFT algorithm is

$$T_{RFFT} = O(n \cdot (\text{number of dipaths with the property } \pi))$$

The algorithm of Rao and Murti (1969) is the fastest of the three circuit vector space algorithms known. The time bound for this algorithm is

$$T_{R\&M} = O(\mu \cdot 2^{\mu}) \qquad (\mu \text{ is nullity of the graph } G)$$

And if μ is proportional to n^2, Char's (1968) master circuit matrix algorithm is faster.

Using a circuit-graph as a guide in finding all circuits, the number of edge-disjoint unions of circuits considered can be reduced. The efficiency of an algorithm for all circuits based on a circuit-graph depends on the efficiency of an algorithm for finding all connected induced subgraphs of a graph. We have proved that there are only four graphs, viz., K_3, K_4, K_4-x and $K_{3,3}$, having all circs as circuits. Thus for any other graph G, a circuit-graph of G can be constructed which is not complete. The number of connected induced subgraphs will be less than 2^{μ}. When a brute-force algorithm is given for connected induced subgraphs, the algorithm for all circuits based on circuit-graph has the same time bound as that of Rao and Murti.

REFERENCES

1. D. Cartwright and T. C. Gleason, "The Number of Paths and Cycles in a Digraph," Psychometrika, 31, 1966, 179-199. MR 33, 906.

2. S. G. Chan and W. T. Chang, "On the Determination of Dicircuits and Dipaths," Proc. Second Hawaii Internat. Conf. System Sciences, 1969, 155-158.

3. J. P. Char, "Master Circuit Matrix," Proc. IEE (London), 115, 6, 1968, 762-770.

4. S. M. Chase, "Analysis of Algorithms for Finding All Spanning Trees of a Graph," Ph.D. Thesis, Report No. 401, Department of Computer Science, University of Illinois at Urbana-Champaign, October 1970.

5. G. H. Danielson, "On Finding the Simple Paths and Circuits in a Graph," IEEE Trans. Circuit Theory, CT-15, 3, 1968, 294-295.

6. N. Deo, "An Introduction to Graph Theory and Its Application to Computer Science and Engineering," Prentice-Hall, Inc., Englewood Cliffs, N. J., 1974 (in press).

7. N. E. Gibbs, "A Cycle Generation Algorithm for Finite Undirected Linear Graphs," J. ACM, 16, 4, 1969, 564-568. CR 19, 833.

8. C. C. Gotlieb and D. G. Corneil, "Algorithms for Finding a Fundamental Set of Cycles for an Undirected Linear Graph," Comm. ACM, 10, 2, 1967, 780-783. CR 14, 355.

9. F. Harary, "Graph Theory," Addison-Wesley, Reading, Mass., 1969.

10. R. C. Holt and E. M. Reingold, "On the Time Required to Detect Cycles and Connectivity in Graphs," Math. Systems Theory, 6, 2, 1972, 103-107.

11. J. Hopcroft and R. Tarjan, "Efficient Algorithms for Graph Manipulation," Comm. ACM, 16, 6, 1973, 372-378.

12. T. Kamae, "A Systematic Method of Finding All Directed Circuits and Enumerating All Directed Paths," IEEE Trans. Circuit Theory, CT-14, 2, 1967, 166-171.

13. R. M. Karp, "Reducibility among Combinatorial Problems" in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher Eds., Plenum Press, New York, 1972.

14. J. W. LaPatra and B. R. Meyers, "Algorithms for Circuit Enumeration," IEEE International Convention Record, Part 1, 12, 1964, 368-373.

15. L. M. Maxwell and G. B. Reed, "Subgraph Identification - Segs, Circuits and Paths," Eighth Midwest Symposium on Circuit Theory (Colorado State University, Fort Collins, Colo.), June 1965, 13-0 - 13-10.

16. P. S. Mei and N. E. Gibbs, "A Planarity Algorithm Based on the Kuratowski Theorem," Proc. AFIPS, Vol. 36, SJCC 1970, 91-93, AFIP Press, N. J.

17. R. L. Norman, "A Matrix Method for Location of Cycles of a Directed Graph," Am. Inst. Chem. Engrs. J., 11, 1965, 450-452.

18. K. Paton, "An Algorithm for Finding a Fundamental Set of Cycles for an Undirected Linear Graph," Comm. ACM, 12, 9, 1969, 514-518. CR 18, 332.

19. K. Paton, "An Algorithm for the Blocks and Cut-Nodes of a Graph," Comm. ACM, 14, 7, 1971, 468-476.

20. M. Prabhaker, "Analysis of Algorithms for Finding all Circuits of a Graph," Thesis, Department of Electrical Engineering, Indian Institute of Technology, Kanpur, India, August 1972.

21. I. Pohl, "A Method for Finding Hamilton Paths and Knight's Tours," Comm. ACM, 10, 7, 1967, 446-449. CR 15, 026.

22. J. Ponstein, "Self-Avoiding Paths and Adjacency Matrix of a Graph," SIAM J. Appl. Math., 14, 1966, 600-609. MR 34, 17.

23. V. V. B. Rao and V. G. K. Murti, "Enumeration of All Circuits of a Graph," Proc. IEEE, 57, 4, 1969, 700-701.

24. V. V. B. Rao, K. Sankara Rao, P. Sankaran and V. G. K. Murti, "Planar Graphs and Circuits," Matrix and Tensor Quarterly, 18, 1968, 88-91.

25. S. M. Roberts and B. Flores, "Systematic Generation of Hamiltonian Circuits," Comm. ACM, 9, 9, 1966, 690-694. MR 33, 672.

26. J. C. Tiernan, "An Efficient Search Algorithm to Find the Elementary Circuits of a Graph," Comm. ACM, 13, 12, 1970, 722-726. CR 21, 083.

27. S. H. Unger, "GIT - A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism," Comm. ACM, 7, 1, 1964, 26-34.

28. H. Weinblatt, H., "A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph," J. ACM, 19, 1, 1972, 43-56.

29. J. T. Welch, "Cycle Algorithms for Undirected Linear Graphs and Some Immediate Applications," Proc. 1965 ACM Natnl. Conf., P-65, 296-301. CR 9, 084.

30. J. T. Welch, "A Mechanical Analysis of the Cyclic Structure of Undirected Linear Graphs," J. ACM, 13, 2, 1966, 205-210. CR 10, 573.

31. T. W. Williams, "Graphs and Their Associated Vector Spaces," Ph.D. Thesis, Colorado State University, 1971. Dissertations Abstracts International, 32, 11, 1972, 6361-B.

32. S. S. Yau, "Generation of all Hamiltonian Circuits, Paths, and Centers of a Graph, and Related Problems," IEEE Trans. Circuit Theory, CT-14, 1, 1967, 79-81.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-73-585	2.	3. Recipient's Accession No.
4. Title and Subtitle ON ALGORITHMS FOR FINDING ALL CIRCUITS OF A GRAPH			5. Report Date June 1973 6.
7. Author(s) Prabhaker Mateti and Narsingh Deo			8. Performing Organization Rept. No.
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF GJ-31222
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C.			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

An efficiency analysis of all known algorithms for finding all circuits of a graph is made, and bounds on the computation time are derived. Best algorithms are exhibited. The vector-space method of circuit finding is discussed at length. Proven is the fact that the graphs K_3, K_4, K_4-x and $K_{3,3}$ are the only ones which have 2^{μ}-1 circuits (μ is the nullity of the graph), or, equivalently, have no two edge-disjoint circuits. A class of graphs, circuit-graphs, derivable out of a graph are defined. If an efficient algorithm for the enumeration of all connected, induced subgraphs is known, these circuit-graphs can advantageously be used in the generation of circuits.

17. Key Words and Document Analysis. 17a. Descriptors

Analysis of algorithms, graph theory, algorithms, circuits, dicircuits, circuit vector space, adjacency matrix, graph search, circuit-graph.

CR category: 5.32

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page) UNCLASSIFIED	22. Price